PRAKTIKUM PEMROGRAMAN JARINGAN
JOBSHEET 9
"Socket Programming"

DOSEN PENGAMPU:
Randi Proska Sandra, M.Sc.

OLEH:
Muhammad Alfarobi
23343011

PROGRAM STUDI INFORMATIKA
DEPARTEMEN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2025

LATIHAN

1. Buatlah sebuah folder baru dengan nama ‘ruangobrol’. Berikut adalah struktur file dan folder dalam folder tersebut.

```
$ mkdir ruangngobrol
```

ol\package.json:

```
{
  "name": "ruangngobrol",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
```

2. Pada file index.js di folder src. Masukanlah source code berikut ini

```
const path = require('path')
const http = require('http')
const express = require('express')
const socketio = require('socket.io')
const Filter = require('bad-words')
const { generateMessage, generateLocationMessage } =
require('./utils/messages')
const { tambahPengguna, hapusPengguna, ambilPengguna, ambilPenggunaDariRoom }
=
require('./utils/users')

const app = express()
const server = http.createServer(app)
const io = socketio(server)

const port = process.env.PORT || 3000;
const publicDirectoryPath = path.join(__dirname, '../public')

app.use(express.static(publicDirectoryPath))

io.on('connection', (socket) => {
    console.log('New WebSocket connection')

    socket.on('join',(options, callback) =>{
        const {error, user} = tambahPengguna({id: socket.id, ...options})
```

```
        if(error) {
            return callback(error)
        }

        socket.join(user.room)
        socket.emit('pesan', generateMessage('Admin','Selamat datang!'))
        socket.broadcast.to(user.room).emit('pesan',
        generateMessage('Admin',`${user.username} telah bergabung`))

        io.to(user.room).emit('roomData', {
            room: user.room,
            users: ambilPenggunaDariRoom(user.room)
        })
        callback()
    })

    socket.on('kirimPesan', (pesan, callback) => {
        const user = ambilPengguna(socket.id)
        const filter = new Filter()

        if(filter.isProfane(pesan)) {
            return callback('Pesan tidak boleh mengandung kata kasar')
        }

        io.to(user.room).emit('pesan', generateMessage(user.username, pesan))
        callback()
    })

    socket.on('kirimLokasi', (coords, callback) => {
        const user = ambilPengguna(socket.id)

        io.to(user.room).emit(
            'locationMessage',
            generateLocationMessage(
                user.username,
                `https://www.google.com/maps?q=${coords.latitude},${coords.lon
gitude}`
            )
        )
        callback()
    })

    socket.on('disconnect', () => {
        const user = hapusPengguna(socket.id)

        if(user) {
            io.to(user.room).emit(
                'pesan',
```

```
                generateMessage(
                    'Admin',
                    `${user.username} telah keluar`
                )
            )

            io.to(user.room).emit('roomData', {
                room: user.room,
                users: ambilPenggunaDariRoom(user.room)
            })
        }
    })
})

server.listen(port, () => {
    console.log(`Server is running on port ${port}!`)
})
```

3. Pada file users.js di folder src/utils, masukanlah source code berikut ini

```
const users = []

//tambahPengguna
const tambahPengguna = ({id, username, room}) => {
    //clean the data
    username = username.trim().toLowerCase()
    room = room.trim().toLowerCase()

    //validate the data
    if(!username || !room) {
        return {error: 'Username dan room dibutuhkan!'}
    }

    //check for exisiting user
    const existingUser = users.find((user) => {
        return user.room === room && user.username === username
    })

    //validate username
    if (existingUser) {
        return { error: 'Username sudah digunakan!'}
    }

    //store user
    const user = {id, username, room}
    users.push(user)
    return {user}
}

//hapusPengguna
```

```
const hapusPengguna = (id) => {
    const index = users.findIndex((user) => user.id === id)
    if(index !== -1) {
        return users.splice(index, 1)[0]
    }
}

//ambilPengguna
const ambilPengguna = (id) => {
    return users.find((user) => user.id === id)
}

//ambilPenggunaDariRoom
const ambilPenggunaDariRoom = (room) => {
    room = room.trim().toLowerCase()
    return users.filter((user) => user.room === room)
}

module.exports = {
tambahPengguna,
hapusPengguna,
ambilPengguna,
ambilPenggunaDariRoom
}
```

4. Pada file messages.js di folder src/utils, masukanlah source code berikut ini

```
const generateMessage = (username, text) =>{
    return {
        username,
        text,
        createdAt: new Date().getTime()
    }
}

const generateLocationMessage = (username, url) => {
    return {
        username,
        url,
        createdAt: new Date().getTime()
    }
}

module.exports = {
    generateMessage,
    generateLocationMessage
}
```

5. Pada file index.html di folder public, masukanlah source code berikut ini

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>kitangobrol</title>
    <link rel="icon" href="/img/favicon.png">
    <link rel="stylesheet" href="/css/styles.min.css">
</head>
<body>
    <div class="centered-form">
        <div class="centered-form__box">
            <h1>ruangobrol</h1>
            <form action="/chat.html">
                <label>Display name</label>
                <input type="text" name="username" placeholder="Display Name"
required>

                <label>Room</label>
                <input type="text" name="room" placeholder="Room" required>

                <button>Bergabung ke Chat</button>
            </form>
        </div>
    </div>
</body>
</html>
```

6. Pada file chat.html di folder public, masukanlah source code berikut ini

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ruangobrol</title>
    <link rel="icon" href="/img/favicon.png">
    <link rel="stylesheet" href="/css/styles.min.css">
</head>
<body>
    <div class="chat">
        <div id="sidebar" class="chat__sidebar"></div>

        <div class="chat__main">
            <div id="messages" class="chat__messages"></div>
            <div class="compose">
                <form id="form-pesan">
                    <input name="pesan" placeholder="Pesan" required
autocomplete="off">
                    <button>Kirim</button>
                </form>
```

```
                <button id ="kirim-lokasi">Share Lokasi</button>
            </div>
        </div>
    </div>

    <script id="message-template" type ="text/html">
        <div class="message">
            <p>
                <span class="message__name">{{username}}</span>
                <span class="message__meta">{{createdAt}}</span>
            </p>
            <p>{{message}}</p>
        </div>
    </script>

    <script id="locationMessage-template" type ="text/html">
        <div class="message">
            <p>
                <span class="message__name">{{username}}</span>
                <span class="message__meta">{{createdAt}}</span>
            </p>
            <p><a href="{{url}}" target="_blank">Lokasi saya sekarang</a></p>
        </div>
    </script>

    <script id="sidebar-template" type ="text/html">
        <h2 class="room-title">Ruang: {{room}}</h2>
        <h3 class="list-title">Anggota Ruang</h3>
        <ul class="users">
            {{#users}}
                <li>- {{username}}</li>
            {{/users}}
        </ul>
    </script>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.0.1/mustache.min.js"
></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.22.2/moment.min.js"></
script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/qs/6.6.0/qs.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/js/chat.js"></script>

</body>
</html>
```

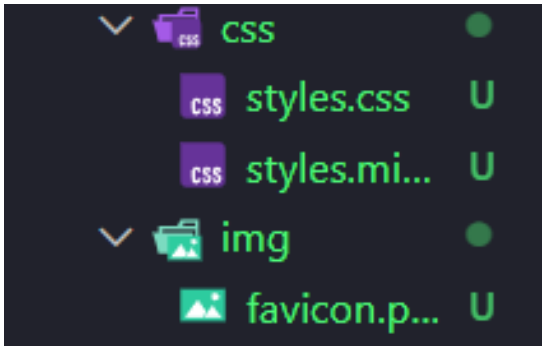
7. Pada file chat.js di folder public/js, masukanlah source code berikut ini

```
const socket = io()

// Elements
const $messageForm = document.querySelector('#form-pesan')
const $messageFormInput = document.querySelector('input')
const $messageFormButton = document.querySelector('button')
const $sendLocationButton = document.querySelector('#kirim-lokasi')
const $messages = document.querySelector('#messages')

// Templates
const messageTemplate = document.querySelector('#message-template').innerHTML
const locationMessageTemplate =
document.querySelector('#locationMessagetemplate').innerHTML
const sidebarTemplate = document.querySelector('#sidebar-template').innerHTML

// Options
const {username, room} = Qs.parse(location.search, {ignoreQueryPrefix: true})

const autoScroll = () => {
    // New message element
    const $newMessage = $messages.lastElementChild

    // Height of the new message
    const newMessageStyles = getComputedStyle($newMessage)
    const newMessageMargin = parseInt(newMessageStyles.marginBottom)
    const newMessageHeight = $newMessage.offsetHeight + newMessageMargin

    //Visible Height
    const visibleHeight = $messages.offsetHeight

    //Height of messages container
    const containerHeight = $messages.scrollHeight

    //How far have I scrolled/
    const scrollOffset = $messages.scrollTop + visibleHeight
    if(containerHeight - newMessageHeight <= scrollOffset) {
        $messages.scrollTop = $messages.scrollHeight
    }
}

socket.on('pesan', (message) => {
    console.log(message)
    const html = Mustache.render(messageTemplate, {
        username: message.username,
        message: message.text,
        createdAt: moment(message.createdAt).format('H:mm')
    })
    $messages.insertAdjacentHTML('beforeend', html)
```

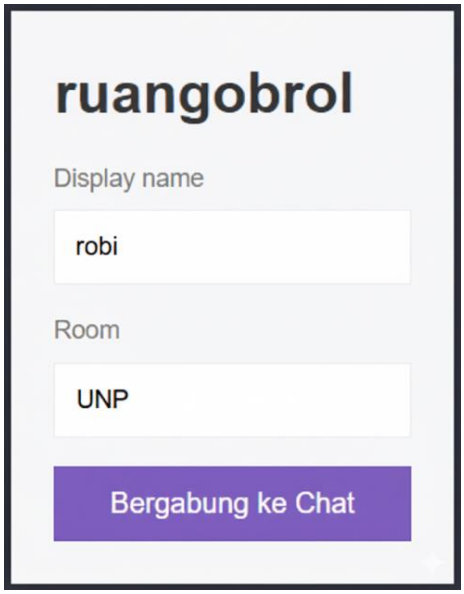
```
    autoScroll()
})

socket.on('locationMessage', (message) => {
    console.log(message)
    const html = Mustache.render(locationMessageTemplate, {
        username: message.username,
        url: message.url,
        createdAt: moment(message.createdAt).format('H:mm')
    })
    $messages.insertAdjacentHTML('beforeend', html)
})

socket.on('roomData', ({room, users}) => {
    const html = Mustache.render(sidebarTemplate, {
        room,
        users
    })
    document.querySelector('#sidebar').innerHTML = html
})

$messageForm.addEventListener('submit', (e) => {
    e.preventDefault()
    //disable form button
    $messageFormButton.setAttribute('disabled', 'disabled')

    const pesan = e.target.elements.pesan.value
    socket.emit('kirimPesan', pesan, (error) => {
        //enable form button
        $messageFormButton.removeAttribute('disabled')
        $messageFormInput.value = ''
        $messageFormInput.focus()

        if(error) {
            return console.log(error)
        }
        console.log('Pesan berhasil dikirim')
    })
})

$sendLocationButton.addEventListener('click', (e) => {
    if(!navigator.geolocation) {
        return alert('Browser anda tidak mendukung Geolocation')
    }

    $sendLocationButton.setAttribute('disabled', 'disabled')

    navigator.geolocation.getCurrentPosition((position) => {
```

```
        socket.emit('kirimLokasi', {
            latitude: position.coords.latitude,
            longitude: position.coords.longitude
        }, () => {
            $sendLocationButton.removeAttribute('disabled')
            console.log('Lokasi berhasil dikirim')
        })
    })
})

socket.emit('join', {username, room}, (error) => {
    if(error) {
        alert(error)
        location.href = '/'
    }
})
```

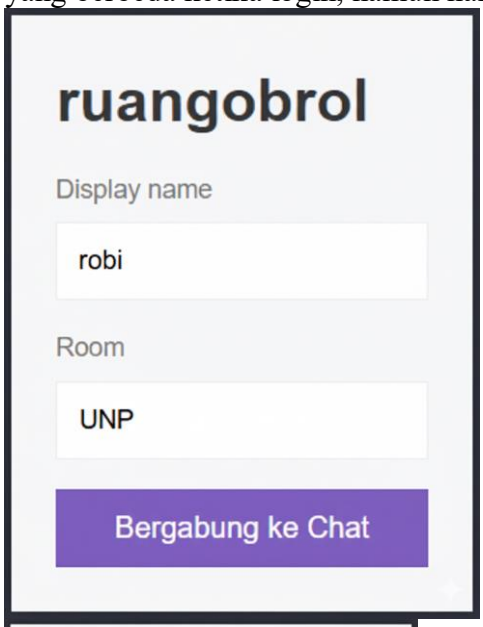
8. Silakan download file berikut untuk file-file yang terkait folder css dan img.

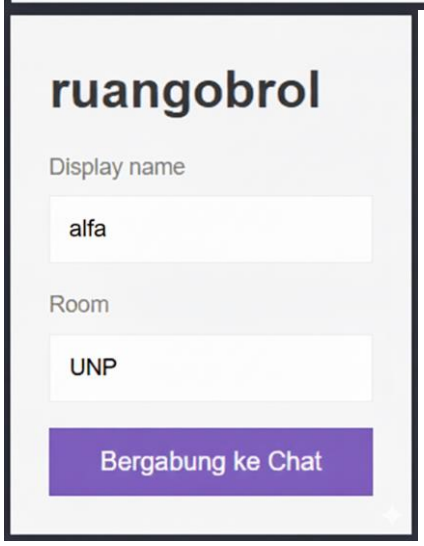


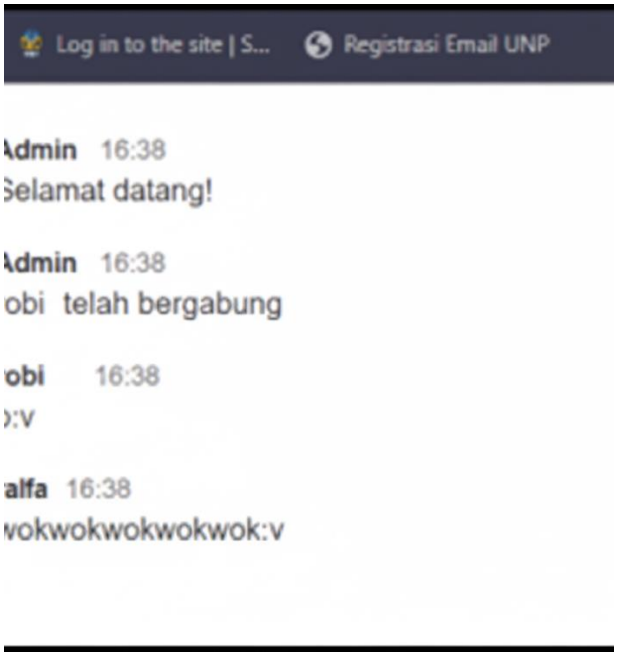
9. Perintah untuk menjalankan aplikasi adalah npm run dev
10. PENTING! Sebelum menjalankan aplikasi, perlu diperhatikan bahwa anda harus menginstall berbagai library yang dibutuhkan, perhatikan source code pada file index.js di folder src. Salah satu library utama adalah socket.io https://www.npmjs.com/package/socket.io
11. Jika telah selesai dan aplikasi berjalan seperti gambar berikut ini, maka lakukanlah commit dan deploy aplikasi anda ke GitHub. Perhatikan langkah-langkah pada jobsheet sebelumnya!



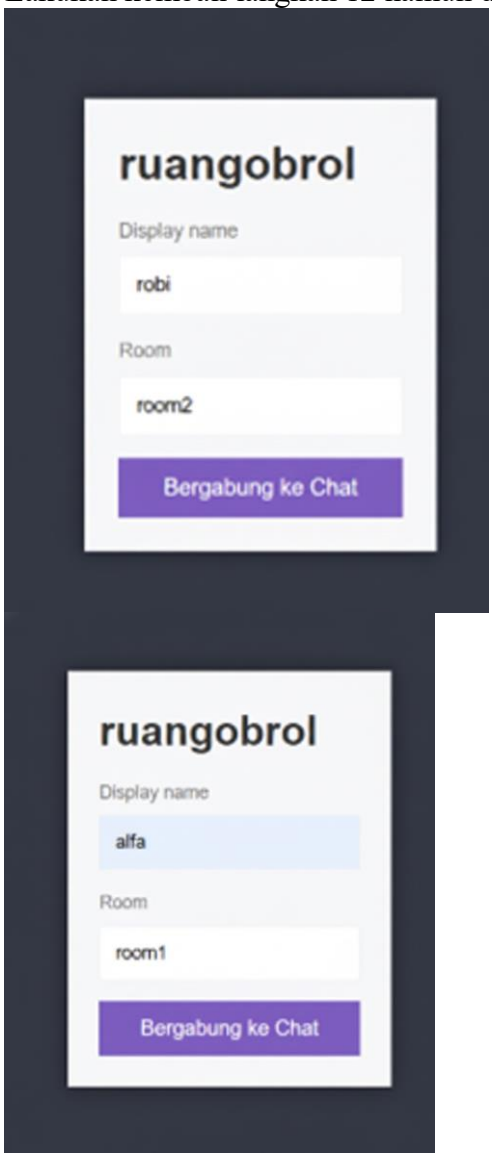
12. Cobalah untuk menjalankan aplikasi dengan mengetikkan localhost:3000 pada dua tab berbeda atau melalui browser berbeda seperti gambar berikut ini!. Gunakan display name yang berbeda ketika login, namun nama room tetap sama.

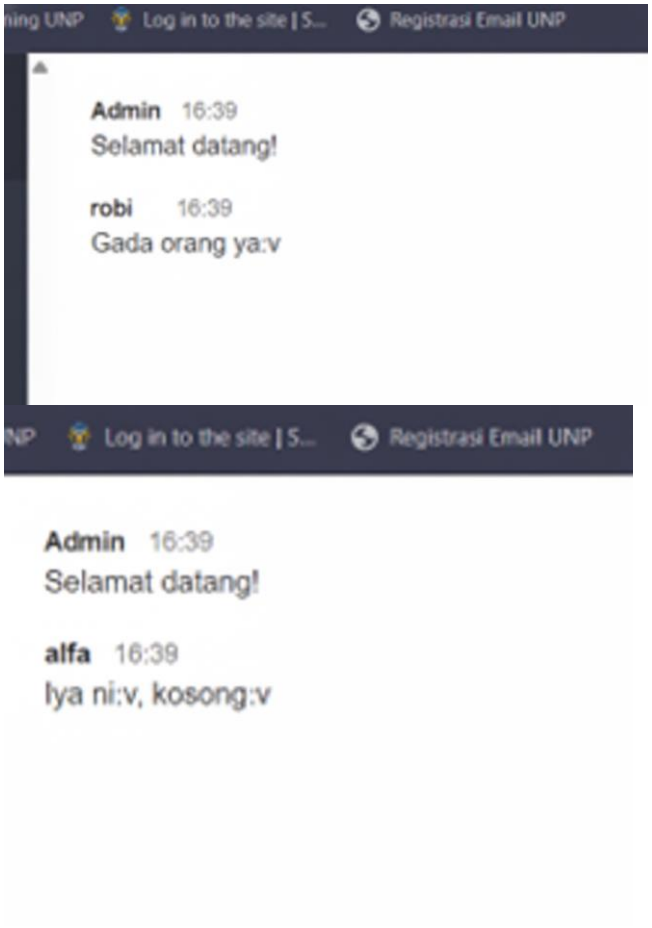






13. Lakukan kembali langkah 12 namun dengan nama room yang berbeda





TUGAS:

1. Jelaskan dengan disertai penjelasan baris kode terkait perbedaan fungsi socket.on yang ada pada file index.js di folder src dan file chat.js pada folder public/js!
   Jawab:

   Pada file index.js yang berada di folder src, fungsi socket.on digunakan di sisi server sebagai mekanisme utama untuk menerima dan menangani event yang dikirim oleh client. Contohnya terlihat pada kode socket.on('join', (options, callback) => { ... }), di mana server akan menunggu event join saat pengguna pertama kali masuk ke aplikasi chat. Setelah event ini diterima, server menjalankan beberapa proses penting, seperti memvalidasi data pengguna, memasukkan pengguna ke dalam room tertentu menggunakan socket.join(user.room), mengirim pesan sambutan langsung ke pengguna dengan socket.emit(...), serta memberi notifikasi kepada pengguna lain di room yang sama melalui socket.broadcast.to(user.room).emit(...). Dari sini terlihat bahwa socket.on di server berperan aktif dalam mengatur alur komunikasi dan kondisi aplikasi.

   Selain event join, socket.on di server juga digunakan untuk menangani pengiriman pesan, misalnya pada socket.on('kirimPesan', (pesan, callback) => { ... }). Pada bagian ini, server menerima pesan dari client lalu memprosesnya terlebih dahulu, seperti melakukan penyaringan kata tidak pantas menggunakan library bad-words. Setelah pesan dinyatakan valid, server meneruskannya ke seluruh pengguna di room yang sama menggunakan io.to(user.room).emit(...). Hal ini menunjukkan bahwa socket.on di sisi server tidak hanya berfungsi sebagai penerima data, tetapi juga menjalankan logika bisnis, validasi, serta pengelolaan distribusi pesan antar pengguna.

   Sebaliknya, pada file chat.js yang berada di folder public/js, fungsi socket.on digunakan di sisi client (browser). Perannya lebih difokuskan pada menerima data yang dikirim oleh

server dan menampilkannya ke antarmuka pengguna. Contohnya dapat dilihat pada socket.on('pesan', (message) => { ... }), di mana client menerima objek pesan dari server, kemudian memformat dan menampilkannya ke halaman chat menggunakan template Mustache. Pada sisi ini, client tidak melakukan validasi atau pengolahan logika yang rumit, melainkan hanya memastikan pesan tampil dengan rapi dan sesuai.

Penggunaan socket.on di client juga terlihat pada event socket.on('roomData', ({ room, users }) => { ... }). Event ini digunakan untuk menerima informasi terbaru tentang nama room dan daftar pengguna yang sedang aktif. Data tersebut kemudian langsung ditampilkan di sidebar menggunakan manipulasi DOM. Dari sini dapat disimpulkan bahwa socket.on di sisi client bersifat lebih pasif, karena hanya menerima hasil pemrosesan dari server dan menyajikannya ke pengguna, tanpa terlibat dalam pengambilan keputusan atau pengelolaan data utama aplikasi.Secara ringkas, perbedaan utamanya adalah: socket.on di server berfungsi untuk menerima event dan mengatur logika serta alur komunikasi aplikasi, sedangkan socket.on di client berfungsi untuk menerima data dari server dan menampilkannya ke antarmuka pengguna.

Aspek Perbandingan	index.js (Server)	chat.js (Client)
Lokasi kode	Backend (Node.js)	Frontend (Browser)
Fungsi utama	Mengelola logika aplikasi chat	Menampilkan data ke pengguna
Sumber event	Event dikirim dari client	Event dikirim dari server
Contoh event	join, kirimPesan, kirimLokasi, disconnect	pesan, locationMessage, roomData
Peran socket.on	Menerima dan memproses data	Menerima dan menampilkan data
Fokus kerja	Logika, validasi, dan distribusi pesan	Tampilan (UI) dan interaksi pengguna

2. Pada saat anda melakukan proses chat seperti pada langkah 12 dan 13. Bukalah inspect pada browser anda. Lalu bukalah menu console. Lakukanlah proses chat dan investigasi apa yang ditampilkan pada console tersebut. Uraikan penjelasan anda dengan mengaitkannya ke baris kode yang menurut anda berhubungan dengan hal tersebut!
Jawab:

Saat proses chat berlangsung dan menu Inspect ke Console dibuka, akan terlihat bahwa setiap kali pesan dikirim atau diterima, console menampilkan sebuah object JavaScript yang berisi informasi pesan. Object ini memuat data seperti nama pengguna, isi pesan, dan waktu pengiriman. Hal tersebut terjadi karena di sisi client terdapat proses yang secara sengaja mencetak data pesan ke console setiap kali event pesan diterima dari server.

Karena server mengirimkan pesan dalam bentuk object, maka setiap kali client menerima data tersebut, isi object akan langsung muncul di console. Inilah alasan mengapa terlihat output seperti data berisi username tertentu, teks pesan, serta waktu pengiriman. Tampilan ini menunjukkan bahwa alur pengiriman pesan dari server ke client berjalan dengan baik dan data diterima secara utuh.

Jika dilihat dari sisi server, object pesan tersebut dibentuk sebelum dikirim ke client. Server mengemas informasi pesan mulai dari pengirim, isi pesan, hingga waktu pengiriman ke dalam satu object, lalu mendistribusikannya ke seluruh pengguna yang berada dalam room yang sama. Object inilah yang kemudian diterima oleh browser dan ditampilkan di console sebagai hasil cetakan data.

Selain menampilkan object pesan, console juga memperlihatkan tulisan seperti “Pesan berhasil dikirim” setiap kali pengguna mengirim pesan. Tampilan ini berasal dari sisi client sebagai respons setelah server selesai menerima dan memproses pesan tanpa kendala.

Dengan kata lain, pesan tersebut menjadi penanda bahwa komunikasi antara client dan server telah berjalan sukses.

Berdasarkan hal tersebut, dapat disimpulkan bahwa tampilan di console berfungsi sebagai alat debugging. Object yang muncul menunjukkan data pesan yang dikirim dari server ke client, sedangkan pesan teks menandakan bahwa proses pengiriman dan respons antara client dan server melalui Socket.IO berhasil dilakukan dengan benar.

3. Pada file chat.html dibagian akhir pada baris kode <script> terdapat penggunaan library mustache, moment dan qs. Jelaskan bagaimana ketiga library ini berfungsi dalam aplikasi yang anda buat, kaitkanlah dengan baris kode yang menurut anda berhubungan!
Jawab:

Pada bagian akhir file chat.html, terdapat beberapa library tambahan yang digunakan agar tampilan dan alur chat menjadi lebih rapi dan mudah digunakan, yaitu Mustache, Moment, dan Qs. Ketiga library ini bekerja di sisi client (browser) dan saling melengkapi dalam menampilkan data chat yang dikirim dari server.

Library Mustache digunakan untuk mengatur tampilan pesan chat, pesan lokasi, dan daftar anggota room tanpa harus menulis HTML secara manual lewat JavaScript. Hal ini terlihat dari adanya template HTML seperti message-template, locationMessage-template, dan sidebar-template yang berisi placeholder seperti {{username}}, {{message}}, dan {{createdAt}}. Placeholder ini nantinya akan diisi data asli menggunakan Mustache di file chat.js. Baris kode yang menunjukkan penggunaan Mustache adalah:

```
socket.on('pesan', (message) => {
    console.log(message)
    const html = Mustache.render(messageTemplate, {
        username: message.username,
        message: message.text,
        createdAt: moment(message.createdAt).format('H:mm')
    })
    $messages.insertAdjacentHTML('beforeend', html)
    autoScroll()
})
```

Dengan bantuan Mustache, data pesan dari server bisa langsung dimasukkan ke template HTML, sehingga kode menjadi lebih rapi dan mudah dibaca dibandingkan harus membuat elemen HTML satu per satu.

Library Moment berfungsi untuk mengatur dan memformat waktu agar tampil lebih enak dilihat oleh pengguna. Data waktu (createdAt) yang dikirim server sebenarnya berupa timestamp, yang kalau ditampilkan mentah akan sulit dipahami. Oleh karena itu, Moment digunakan untuk mengubahnya menjadi format jam dan menit. Hal ini terlihat pada baris kode berikut: createdAt: moment(*message*.createdAt).format('H:mm').

Dengan kode ini, waktu pesan yang tadinya berupa angka besar diubah menjadi format sederhana seperti 16:54, sehingga pengguna bisa langsung tahu kapan pesan dikirim.

Sementara itu, library Qs digunakan untuk mengambil data username dan room dari URL saat pengguna masuk ke halaman chat. Ketika pengguna masuk ke halaman seperti /chat.html?username=panca&room=room1, data tersebut perlu dibaca oleh JavaScript. Proses ini dilakukan dengan Qs pada baris kode: const {username, room} = Qs.parse(location.search, {ignoreQueryPrefix: true}).

Kode ini membuat aplikasi bisa mengetahui siapa username pengguna dan berada di room mana, tanpa harus mengetik ulang. Data ini kemudian dikirim ke server melalui event join agar server bisa mengelola pengguna dan room dengan benar. Secara

keseluruhan, Mustache bertugas mengatur tampilan, Moment mengatur waktu pesan, dan Qs mengatur data dari URL. Kombinasi ketiga library ini membuat aplikasi chat yang dibuat menjadi lebih rapi, informatif, dan nyaman digunakan oleh pengguna.

4. Bukalah chat.js, dan perhatikan bahwa ada beberapa baris kode yang telah ditandai dengan komentar elements, templates dan options. Jelaskan baris kode tersebut dan bagaimana kode tersebut berhubungan dengan file chat.html dan file index.html!
   Jawab:

   Pada file chat.js, bagian yang diberi komentar Elements berfungsi untuk menghubungkan JavaScript dengan elemen-elemen HTML yang ada di chat.html. Contohnya, variabel $messageForm digunakan untuk mengambil elemen <form id="form-pesan">, $messageFormInput mengarah ke kolom input pesan, dan $messageFormButton digunakan untuk mengatur tombol kirim, misalnya untuk menonaktifkannya sementara saat pesan sedang diproses. Selain itu, $sendLocationButton terhubung dengan tombol Share Lokasi, sedangkan $messages menunjuk ke elemen yang menjadi wadah utama untuk menampilkan seluruh pesan chat. Dengan pengambilan elemen-elemen ini, JavaScript dapat merespons aksi pengguna secara langsung tanpa perlu memuat ulang halaman.

   Pada bagian Templates, aplikasi mengambil isi template HTML menggunakan variabel seperti messageTemplate, locationMessageTemplate, dan sidebarTemplate. Template-template ini berasal dari elemen <script> khusus di chat.html dan berfungsi sebagai pola tampilan pesan teks, pesan lokasi, serta informasi sidebar. Saat event socket.on('pesan', ...) atau event lain diterima, data dari server akan diproses menggunakan Mustache.render(...) sehingga nilai seperti message.username, message.text, dan message.createdAt dapat ditampilkan sesuai format yang sudah ditentukan. Dengan pendekatan ini, tampilan pesan menjadi konsisten dan mudah dikelola.

   Selanjutnya pada bagian Options, data username dan room diambil dari URL menggunakan Qs.parse(location.search, ...). Contohnya, ketika URL berisi ?username=panca&room=room1, nilai tersebut otomatis diproses dan disimpan dalam variabel username dan room. Data ini kemudian dikirim ke server melalui event socket.emit('join', { username, room }, ...). Di sisi server, event join ini digunakan untuk mendaftarkan pengguna ke room yang sesuai, mengirim pesan sambutan, serta memperbarui daftar anggota room.

   Secara keseluruhan, bagian Elements berfokus pada pengambilan dan pengendalian elemen HTML, Templates mengatur bagaimana data ditampilkan menggunakan Mustache, dan Options berperan sebagai penghubung antara data dari URL, logika di client (chat.js), dan manajemen pengguna di server (index.js). Ketiganya bekerja bersama agar aplikasi chat dapat berjalan secara real-time, terstruktur, dan responsif.

5. Jelaskan fungsi file messages.js dan users.js dan bagaimana baris kode pada kedua file ini terhubung dengan index.js, chat.js dan kedua file html (chat.html dan index.html)
   Jawab:

   Menurut saya, messages.js berfungsi sebagai pembuat format pesan yang akan dikirim dari server ke client. Di dalam file ini terdapat fungsi generateMessage dan generateLocationMessage yang tugasnya membungkus data pesan menjadi satu object yang rapi dan konsisten. Misalnya generateMessage menerima username dan text, lalu otomatis menambahkan createdAt sebagai penanda waktu. Object inilah yang dikirim oleh server di index.js melalui event socket.emit('pesan', ...) atau io.to(user.room).emit('pesan', ...), kemudian diterima oleh client di chat.js melalui socket.on('pesan', ...) dan ditampilkan

ke chat.html menggunakan template Mustache. Dengan adanya file ini, format data pesan antara server dan client selalu seragam dan mudah diolah.

Masih di messages.js, fungsi generateLocationMessage bekerja dengan konsep yang sama, hanya saja isi datanya berupa url lokasi. Fungsi ini dipakai saat server menerima event pengiriman lokasi, lalu mengirimkan object lokasi ke client melalui event locationMessage. Di sisi client, data tersebut diterima oleh socket.on('locationMessage', ...) dan dirender menggunakan template locationMessage-template. Menurut saya, pemisahan ini membuat kode server jauh lebih rapi karena tidak perlu menulis ulang struktur pesan setiap kali mengirim data.

Berbeda dengan itu, users.js berfungsi sebagai pengelola data pengguna yang sedang aktif. File ini menyimpan data user dalam array users dan menyediakan fungsi seperti tambahPengguna, ambilPengguna, hapusPengguna, dan ambilPenggunaDariRoom. Fungsi tambahPengguna dipanggil saat event join terjadi di index.js, dengan data username dan room yang dikirim dari chat.js melalui socket.emit('join', ...). Fungsi ini bertugas merapikan input, melakukan validasi, dan memastikan tidak ada username yang sama dalam satu room.

Fungsi lain seperti ambilPengguna digunakan server untuk mengetahui siapa pengirim pesan saat event kirimPesan, sedangkan hapusPengguna dipanggil ketika user disconnect agar data tetap sinkron. Data pengguna dalam satu room kemudian dikirim ke client melalui event roomData dan ditampilkan di sidebar chat.html menggunakan sidebar-template. Dengan mekanisme ini, daftar anggota room selalu ter-update secara real-time.

Kesimpulannya, messages.js fokus mengatur struktur data pesan, users.js fokus mengelola data pengguna, index.js menjadi pusat logika server, chat.js menangani interaksi client, dan chat.html/index.html berperan sebagai tampilan. Menurut saya, pemisahan tugas seperti ini membuat alur aplikasi jelas, terstruktur, dan mudah dikembangkan ke tahap selanjutnya.

6. Bagaimana aplikasi ini bisa mengirimkan lokasi? Jelaskan apa yang terjadi dengan disertai penjelasan baris kode!

Jawab:

Aplikasi ini dapat mengirimkan lokasi karena memanfaatkan fitur Geolocation bawaan browser yang kemudian dikirim menggunakan Socket.IO. Prosesnya dimulai dari sisi client di file chat.js saat saya menekan tombol “Share Lokasi”, yang ditangani oleh event $sendLocationButton.addEventListener('click', ...).

Saat tombol ditekan, aplikasi terlebih dahulu mengecek dukungan browser melalui navigator.geolocation. Jika fitur ini tidak tersedia, maka proses langsung dihentikan dan pengguna diberi peringatan. Jika tersedia, browser akan meminta izin lokasi, lalu mengambil koordinat melalui navigator.geolocation.getCurrentPosition(...) yang berisi position.coords.latitude dan position.coords.longitude.

Koordinat tersebut kemudian dikirim ke server menggunakan socket.emit('kirimLokasi', { latitude, longitude }, callback). Event kirimLokasi ini menjadi penanda bahwa client ingin membagikan lokasi ke pengguna lain.Di sisi server (index.js), event ini ditangani oleh socket.on('kirimLokasi', (coords, callback) => { ... }). Server terlebih dahulu mengambil data user dengan ambilPengguna(socket.id) dari users.js, lalu membentuk pesan lokasi menggunakan generateLocationMessage(...) dari messages.js. Koordinat coords.latitude dan coords.longitude diubah menjadi link Google Maps dengan format https://www.google.com/maps?q=latitude,longitude, kemudian dikirim ke seluruh pengguna dalam room yang sama menggunakan io.to(user.room).emit('locationMessage', ...).

Di sisi client, pesan lokasi ini diterima melalui socket.on('locationMessage', (message) => { ... }). Data yang diterima berupa message.username, message.url, dan message.createdAt, lalu ditampilkan ke halaman chat menggunakan Mustache.render(locationMessageTemplate, ...) dengan template <script id="locationMessage-template"> di chat.html. Link lokasi akan muncul dengan teks seperti “Lokasi saya sekarang” yang bisa langsung diklik.

Setelah server selesai memproses pengiriman lokasi, callback() dijalankan sehingga tombol Share Lokasi diaktifkan kembali melalui $sendLocationButton.removeAttribute('disabled'). Dengan alur ini, pengiriman lokasi berjalan secara real-time, mulai dari izin lokasi di browser, pengiriman data melalui Socket.IO, pengolahan di server, hingga ditampilkan ke semua pengguna dalam satu room.

7. Kenapa aplikasi ini dijalankan menggunakan perintah npm run dev bukan menggunakan perintah node diikuti nama file seperti pada jobsheet-jobsheet sebelumnya? Coba juga jalankan aplikasi menggunakan perintah npm run start, investigasi apa yang terjadi dan apa yang membedakannya dengan npm run dev?
   Jawab:

   Aplikasi ini saya jalankan menggunakan perintah npm run dev karena pada proyek ini proses menjalankan server sudah diatur melalui script di file package.json, bukan lagi langsung memakai node index.js seperti pada jobsheet sebelumnya. Dengan cara ini, alur menjalankan aplikasi jadi lebih rapi dan praktis karena semuanya sudah “dibungkus” oleh npm.

   Saat saya menjalankan npm run dev, npm akan membaca bagian scripts di package.json dan mengeksekusi perintah yang sudah ditentukan, biasanya menggunakan nodemon src/index.js. Artinya, file index.js tetap dijalankan, tetapi lewat nodemon. Keuntungannya, setiap kali saya mengubah kode baik di index.js, chat.js, maupun file lain server akan otomatis restart tanpa perlu saya hentikan dan jalankan ulang secara manual. Ini sangat membantu saat pengembangan dan debugging aplikasi chat yang berjalan secara real-time.

   Berbeda dengan itu, ketika saya menjalankan npm run start, npm akan menjalankan script start yang umumnya menggunakan node src/index.js. Pada mode ini, server berjalan secara normal tanpa auto-reload. Jadi, jika saya mengubah kode saat server masih berjalan, perubahan tersebut tidak langsung aktif sampai server dimatikan dan dijalankan ulang. Inilah perbedaan yang paling terasa antara npm run dev dan npm run start.

   Kesimpulannya, npm run dev lebih cocok saya gunakan saat proses pengembangan karena lebih efisien dan fleksibel, sedangkan npm run start lebih pas untuk menjalankan aplikasi dalam kondisi siap pakai. Sementara itu, penggunaan node nama_file.js seperti pada jobsheet sebelumnya masih relevan untuk aplikasi sederhana, tetapi untuk proyek yang lebih terstruktur seperti aplikasi chat ini, penggunaan npm script membuat pengelolaan dan eksekusi aplikasi terlihat lebih rapi dan profesional.

8. Selain socket.on, fungsi socket apa lagi yang digunakan dalam aplikasi ini. Silakan telusuri dan jelaskan pendapat anda disertai dengan baris kode!
   Jawab:

   Selain socket.on yang berfungsi untuk menerima event, aplikasi ini juga sangat bergantung pada socket.emit sebagai pengirim data utama dalam komunikasi real-time. Di sisi client (chat.js), saya menggunakan socket.emit('join', { username, room }, callback) untuk mengirim data pengguna ke server saat pertama kali masuk ke halaman chat. Event join ini kemudian diterima oleh server di index.js melalui socket.on('join', ...) untuk mendaftarkan user ke room yang sesuai. Selain itu, socket.emit('kirimPesan', pesan, callback) dipakai untuk mengirim pesan teks, dan socket.emit('kirimLokasi', { latitude,

longitude }, callback) digunakan untuk mengirim data lokasi ke server. Dari sini terlihat jelas bahwa socket.emit berperan sebagai jembatan pengirim data dari client ke server.

Di sisi server, saya juga menggunakan socket.join(user.room) yang fungsinya khusus untuk memasukkan sebuah socket ke dalam room tertentu. Dengan adanya socket.join, server bisa mengelompokkan user berdasarkan room, sehingga pesan hanya dikirim ke pengguna yang berada di room yang sama. Tanpa fungsi ini, semua event akan tersebar ke seluruh client dan konsep ruang chat tidak akan berjalan dengan benar.

Aplikasi ini juga memanfaatkan socket.broadcast.to(user.room).emit(...). Fungsi socket.broadcast digunakan untuk mengirim pesan ke semua user di dalam room kecuali pengirimnya sendiri. Contohnya saat ada user baru yang masuk, server mengirim pesan notifikasi bergabung ke anggota lain di room tersebut, sementara user yang baru masuk tidak menerima pesan itu karena sudah mendapatkan pesan sambutan tersendiri.

Selain itu, saya menggunakan io.to(user.room).emit(...) untuk mengirim event ke semua user dalam satu room, termasuk pengirimnya. Fungsi ini dipakai untuk mengirim pesan chat, pesan lokasi, serta data anggota room melalui event seperti pesan dan roomData. Dengan mekanisme ini, semua user di room selalu menerima informasi yang sama secara sinkron dan real-time.

Terakhir, aplikasi ini memanfaatkan event bawaan Socket.IO yaitu disconnect, yang ditangani melalui socket.on('disconnect', ...). Event ini otomatis terpanggil ketika user menutup tab atau koneksi terputus. Di dalam proses ini, server menjalankan hapusPengguna(socket.id) lalu mengirim notifikasi ke room menggunakan io.to(user.room).emit('pesan', generateMessage('Admin', 'user telah keluar')). Dengan cara ini, saya bisa memastikan bahwa status user dan informasi di room selalu ter-update secara real-time ketika ada anggota yang keluar.

9. Jelaskan terkait ini real-time bidirectional event-based communication disertai penjelasan baris kode sesuai aplikasi yang anda buat!

Jawab:

Menurut saya, konsep real-time bidirectional event-based communication berarti aplikasi mampu melakukan komunikasi dua arah secara langsung antara client dan server dengan berbasis event. Pada aplikasi chat ini, client (browser) dan server (Node.js + Socket.IO) sama-sama bisa mengirim dan menerima event kapan saja, tanpa harus menunggu pola request–response seperti pada HTTP biasa. Hal ini sudah terlihat sejak awal saat koneksi dibuat menggunakan const socket = io() di chat.js, yang otomatis membuka jalur komunikasi dua arah antara browser dan server.

Sifat event-based terlihat jelas dari penggunaan pasangan socket.emit dan socket.on. Ketika saya mengirim pesan, client menjalankan socket.emit('kirimPesan', pesan, callback) di chat.js. Event kirimPesan ini langsung dikirim ke server dan ditangkap oleh socket.on('kirimPesan', ...) di index.js. Setelah diproses, server mengirimkan kembali pesan tersebut ke semua user dalam satu room menggunakan io.to(user.room).emit('pesan', generateMessage(...)). Tanpa perlu refresh halaman, pesan langsung muncul di layar semua pengguna, dan inilah contoh nyata komunikasi berbasis event.

Sifat bidirectional (dua arah) terlihat karena bukan hanya client yang aktif mengirim event ke server, tetapi server juga bisa secara langsung mengirim event ke client. Contohnya saat user baru masuk ke room, server mengirim pesan sambutan menggunakan socket.emit('pesan', generateMessage('Admin', 'Selamat datang!')), lalu memberi notifikasi ke user lain dengan socket.broadcast.to(user.room).emit('pesan', ...). Semua event ini diterima oleh client melalui socket.on('pesan', (message) => { ... }) dan langsung ditampilkan. Ini menunjukkan bahwa server tidak bersifat pasif, melainkan aktif berkomunikasi dengan client.

Sifat real-time terlihat dari respons yang terjadi seketika saat ada perubahan, seperti user bergabung, mengirim pesan, berbagi lokasi, atau keluar dari room. Misalnya pada pengiriman lokasi, client mengirim koordinat melalui socket.emit('kirimLokasi', { latitude, longitude }), lalu server langsung mengubahnya menjadi link Google Maps dan membagikannya menggunakan io.to(user.room).emit('locationMessage', generateLocationMessage(...)). Client lain langsung menerima event tersebut lewat socket.on('locationMessage', ...) tanpa jeda yang terasa.

Kesimpulannya, menurut saya aplikasi chat ini benar-benar menerapkan real-time bidirectional event-based communication karena client dan server saling berkomunikasi dua arah, berbasis event, dan memberikan respons secara instan dalam hitungan milidetik.